Index-Free Logging: Are Indexes Necessary, or Simply Overhead

INDEX-FREE LOGGING & LOG MANAGEMENT

Index-Free Logging & Log Management | The world of log management is ridden with woes because most solutions are based on a classical database technology: the index.

INDEX-FREE LOGGING

The world of log management is ridden with woes because most solutions are based on a classical database technology: the index. Indexing has been an accepted and useful approach to collecting and analyzing log data for well over a decade. But its time has passed. Based on index-free architectures, modern log management systems like Humio, Loki, and Scalyr provide faster service at a much lower cost.

THE INDEXING TRADE-OFF

"Database indexes provide a trade off suitable for systems with low ingest rate and high query frequency. The core activity with logs is to write a lot and only search parts of them in specific time ranges, when an incident occurs. Indexes are good for many things but not for logging."

With indexes, you pay disk space and up-front CPU cost to make queries across the entire dataset faster.

Building an index takes time (CPU cycles as well as wall-clock time), and an index needs disk space.

Indexes are good for many things but not for logging.

If the time and space to build the index grows out of proportion with the real data that you are actually interested in, then you have lost.

Consider a cartographer creating a map – she has to work within certain constraints to produce useful work: (a) the map should be of appropriate scale i.e., producing a map that's bigger than the real world is useless, and (b) it must be finished in a timely fashion to be useful – if realities have changed before the map is done then it is only of historical interest.

The logging use case and the nature of logging, event, and trace data compound to make indexing a bad match for this application.

THE 'HIGH-CARDINALITY' PROBLEM

One of the things making indexes of event data so expensive to prepare and so large in size is the high-cardinality problem. Data has high cardinality when it contains a large set of keys and values.

With log data, user-defined events, and traces, users can often create many arbitrary keys (or columns in database terminology). Those keys often have values that are distinct for individual events. These can be trace and span IDs, user names, IP addresses...anything in the data set, really.

The value domain for these kinds of properties is such that indexing them all often results in indexes that are significantly larger than the data you're trying to put into the system and it takes an unreasonably long amount of time to compute this com prehensive index. Thus it may already be irrelevant by the time it's done.

It would indeed be desirable to

- not spend so much time before we can search the data
- not waste so much disk space.

If this was possible, we could make the system much more useful and inexpensive to operate.

INDEX-FREE LOGGING TO THE RESCUE

Index-free event stores do things differently. Here, we essentially store data in buckets and label the buckets with information letting the query engine determine if some sought-after data could possibly be in that bucket or not.

For Humio's data store, these bucket labels include the time range of the data in the bucket, the source and type of data, and a bloom filter computed over the keys and values of the data stored in the bucket.

When a query arrives, we compare the query to the labels of all the buckets, and decide which buckets are relevant to search. And then we simply search the actual contents of the relevant buckets. That's it. Humio works this way, and other logging solutions are using a similar approach including e.g. Loki.

One could argue that our 'labeling of buckets' is sort of an index, just a coarse grained one. However, the moniker Index-free Logging serves to highlight that we do not index the individual keys and values; the labels only describe the data in aggregate.

Beyond the labeling, this technique also affords an easy additional win by compressing the data in the buckets. Compressed data takes up less space on disk and in the previous page cache. It is faster to move around; both in terms of I/O and memory bandwidth. It is not unusual to see gains in the order of 50–100x for disk requirements when comparing indexing to index-free logging.

MODERN HARDWARE AND STREAMING I/O

The Index-free data store approach also goes hand in hand with the implementation techniques that work well with modern hardware.

Because an index-free data store does brute-force search in each individual bucket of data, the data does not need to be sorted. We can use simple append-to-file operations to write to a bucket and use streaming reads when processing it. Using only sequential data access is an instant win that puts less stress on the hardware and avoids a good deal of I/O waiting.

Modern hardware is optimized for these kinds of program behaviours and it can be counter intuitive that doing extra work this way pays off. Examples of how this works are described in Scalyr's blog on 1TB/s brute force, or Humio's 'How fast can you grep?'.

STREAMING DATA: ALERTS AND DASHBOARDS

Indexes are often assumed necessary for fast dashboards and alerts. And indeed: Database indexes provide a trade off suitable for systems that have low ingest rate and high query frequency

If your dashboards are implemented by querying the database every so often, you would indeed need indexes – because dashboards and alerts need to be updated often and compute in a timely fashion.

With Humio however, we process alerts and dashboards directly in the ingest pipeline, independently of storing the data in the event store. This means the part of the query engine dealing with live data is a separate stream processing engine, but with the exact same query language. This has several profound implications:

It would indeed be desirable to

- Alerts and dashboards are processed in near real-time, as the data is in memory anyway i.e., no additional I/O needed
- Searching the on-disk event store is essentially limited to interactive queries when doing incident response or debugging.

Other solutions use a separate metrics store (such as Prometheus) to service dashboards. Humio is unique in that it provides a pipeline to service both metrics, logs, and events in an integrated fashion.

Removing the workload of running dashboards and alerts from the event store engine workload removes the need to support transactional queries. We only need to service interactively from there. Interactive queries, with real people typing them into a console, tend to be of much lower frequency and often not even concurrent. If the data is appropriately spread around the cluster, then an individual user can ideally get the entire search processing capacity of the cluster for a short time interval.

THE "I WISH I INDEXED THAT" SYNDROME

Log data is by nature volatile and when your system is in trouble you want to find things that you did not think about up-front. Attributes you thought were always numbers may suddenly be strings, or you want to be able to do free text search over the entire body of logs on something you did not index.

If you've ever been in a situation where your schemas or templates failed to match the data over time or you have seen a case where you said,

"I wish I indexed that", then you know what we mean. In a solution based on indexing, it would be necessary to re-index the data or give up. With index-free logging you will not be in this situation because everything is searchable.

SUMMARY

Indexing has its purpose when used in applications other than log management, such as whole text search, databases and web searches. In legacy approaches to log management, indexing was used because it was "close enough" and worked sufficiently. But technology has moved on, and the second wave techniques like Humio's are available that are purpose-built specifically for log management, and thus accomplish two things at once - less cost and better results.

Index-free Logging provides for a different set of tradeoffs uniquely better suited for logs, events, and traces than solutions based on exact term indexing.

The benefits for use cases in this space are many and include:

- lower ingest latency
- near-real time alerts and dashboards
- lower disk space requirements
- much lower hardware requirements

In short – more time to understand and debug your real solution, not troubleshooting the logging platform – fewer headaches and more fun.